# Part 5: ER-Model III

**References:**

- Elmasri/Navathe:Fundamentals of Database Systems, 3rd Edition, 1999.
  Ch. 3, "Data Modeling Using the Entity-Relationship Model"
  Ch. 4, "Enhanced Entity-Relationship and Object Modeling"
- Silberschatz/Korth/Sudarshan: Database System Concepts, 3rd Ed.,
  Ch. 2, "Entity-Relationship Model".
- Ramakrishnan: Database Management Systems, Mc-Graw Hill, 1998,
  Ch. 14, "Conceptual Design and the ER-Model"
- Rauh/Stickel: Konzeptuelle Datenmodellierung (in German), Teubner, 1997.
- Kemper/Eickler: Datenbanksysteme (in German), Ch. 2, Oldenbourg, 1997.
- Teorey: Database Modeling and Design, Third Edition, 1999.
- Barker: CASE*Method, Entity Relationship Modelling, Oracle/Addison-Wesley, 1990.
- Lipeck: Skript zur Vorlesung Datenbanksysteme (in German), Univ. Hannover, 1996.





## Objectives

- Advanced ER-constructs:
  - weak entities
  - optional/multivalued/structured attributes
  - specialization (subclasses).
- Designing ER-diagrams for given applications.
- Comparison and critical evaluation of ER-schemas.
- Sources of information (input) for database design.
- Functionality of design tools (only general impression).





## Overview

1. General Remarks on Extended ER-Models
2. Advanced ER-Constructs I: Weak Entities
3. Advanced ER-Constructs II:
   Optional, Multivalued, Structured Attributes
4. Advanced ER-Constructs III: Specialization
5. Miscellaneous Remarks on DB Design





## Extended ER-Models (1)

- Many extensions to the basic ER-model have been proposed.
- The goals are to capture more meaning from the real world and to simplify the conceptual design.
- The focus is on avoiding additional constraints whenever possible.

  If a situation can be formally expressed with constructs of the model, this is better than adding constraints in natural language. E.g. we can then give a formal translation into the relational model.
- Also, additional constructs may help explain the relation to the real world better (like a comment).

## Extended ER-Models (2)

- Adding more constructs to the ER-model gives Extended ER (EER)-models.
- There are many different proposals for such models.

  Therefore, the notation for the extended constructs varies quite a bit among different authors.
- Company policies and the available CASE tools determine which extensions can be used.
- Most extensions make the translation into the relational model harder, but when they simplify the difficult conceptual design, this compensates for problems during logical design.





## Overview

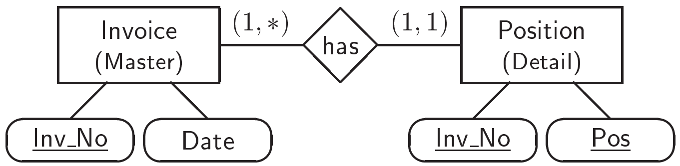
1. General Remarks on Extended ER-Models
2. Advanced ER-Constructs I: Weak Entities
3. Advanced ER-Constructs II:
   Optional, Multivalued, Structured Attributes
4. Advanced ER-Constructs III: Specialization
5. Miscellaneous Remarks on DB Design





## Weak Entities (1)

- An entity may describe a kind of “detail” that cannot exist without a “master” or “owner” entity.
- So there is a relationship with a (1,1) cardinality pointing to the owner.
- In addition the key of the owner entity is inherited and becomes part of the key of the detail entity:







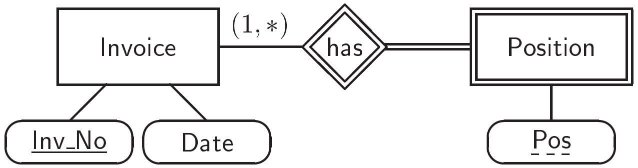

## Weak Entities (2)

- Without a specific construct for this situation, we would need the constraint that
  - if two entities are connected via the relationship “has”,
  - then their attribute “Inv_No” has the same value.

  E.g. invoice 12 cannot have position 2 in invoice 36 as detail.
- Such constraints occur when an entity does not have a key by itself, but is only unique in the context of some other entity.
- So in these cases we always have composed keys:
  - A classroom is identified by a building and a room number.
  - A subexercise is identified by the exercise number (e.g. 1) and a letter (e.g. a).

# Weak Entities (3)

- Weak entities were introduced for such situation where an entity needs to borrow key attributes from a related entity.
- Weak entities are marked in ER-diagrams by doubling their box, the connecting line, and the relationship diamond:




- Only the extension to the borrowed/inherited key is shown. Since it is only a partial key, it is dashed underlined.





# Weak Entities (4)

- So the real key of the weak entity consists of the key attribute of the "owner" entity type (automatically inherited, not explicitly shown) plus the dashed-underlined partial key.
- Another way to look at this is that here the relationship contributes to the identification of the entity, whereas usually only attributes are used for this purpose (in a key).

  The double line can be understood as an "underlined line".
- The cardinality $(1, 1)$ is automatically implied by the weak entity construct. It does not have to be specified explicitly.

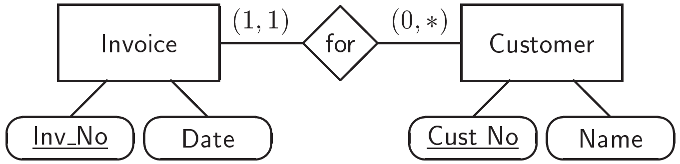
  Other relationships cannot be used: There would be no unique owner entity from which the key value can be inherited.





# Weak Entities (5)

- Note that a $(1, 1)$ cardinality does not automatically mean "weak entity". The entity can still have a key of its own:




- A weak entity is needed only when the key of one entity contains the key of a related entity.

  A weak entity must add further key attributes to the key of its owner. Otherwise use a specialization ($\rightarrow$ below).





# Weak Entities (6)

- Sometimes the master/owner entity is called "parent entity", and the dependent weak entity is called "child entity".
- Entities with their own key are called regular/strong entities.
- Weak entities are normal entities except that their key is constructed in a special way.
- So weak entities can have normal relationships besides the one via which the key is inherited.


- A weak entity can even have a second (normal) relationship to its owner.

# Weak Entities (7)

- Weak entities can be owner entities for other weak entities.

  So you can have an entire hierarchy of parent-child relationships (e.g. "grandchildren").
- Cycles are forbidden: It is impossible that A is a weak entity with owner B, and B is a weak entity with owner A.

Exercise:

- Model a set of online quizzes. Each quiz is identified by a title, each question within a quiz by a number, and each answer to a given question by a letter.
- What is the complete key of each of the three entity types?





# Association Entities (1)

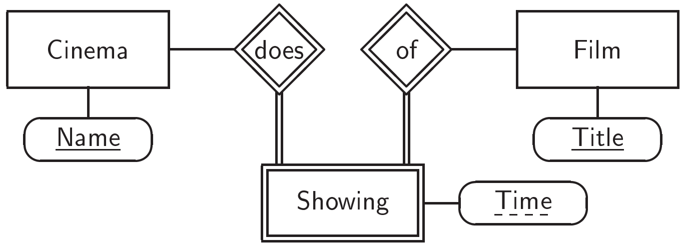
- Reasons for turning a relationship into an entity:
  - The relationship is ternary and the utilized version of the ER-model supports only binary relationships.
  - For some reason, only one-to-many relationships should be used, but the relationship is many-to-many.
  - A relationship between this relationship and some entity type is needed.
  - A relationship-attribute does not have a unique value.
  - The CASE-tool does not support attributes of relationships.
- Then a weak entity is created. It inherits the keys of the entity types that participate in the original relationship.





# Association Entities (2)

- Weak entities can have multiple parents/owners:



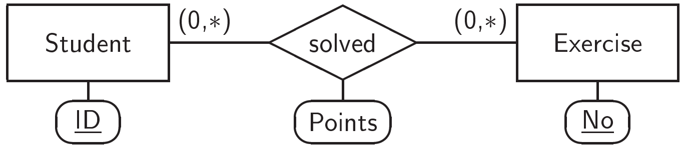

- The key of "Showing" consists of "Name", "Title" and "Time".
- Weak entities with several parents/owners are sometimes called "association entities". ("Weak entity" is equally ok.)
- The owners must have disjoint key attribute names.



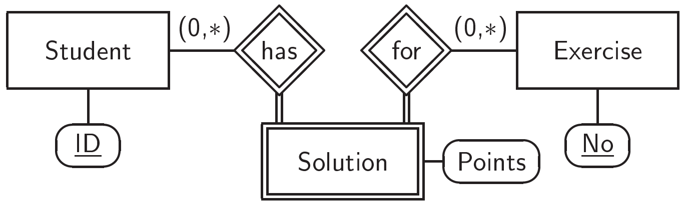


# Association Entities (3)

- Relationship:




- Association Entity (completely equivalent):



## Overview

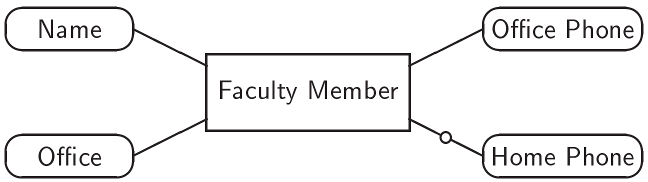






## Optional Attributes (1)

- Sometimes, the value of an attributes is unknown or non-existent.




- E.g. not all faculty members want their home phone number stored, those newly relocated might not have a phone.
- So "Home Phone" is an optional attribute here.





## Optional Attributes (2)

- Undefined attributes correspond to "null values" in the relational model (see below).
- The semantics of optional attributes is quite complicated, partially because a missing value can mean a variety of different things, e.g.:
  - There is a value, but it is not known.
  - There is no value.
  - The attribute makes no sense for this specific entity.

    E.g. a typing speed for a pilot.
- Try to avoid optional attributes whenever possible.





## Optional Attributes (3)

- In practice it is quite common that not all values are available at once.

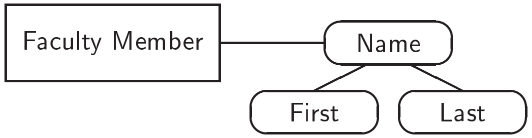
  In such cases allowing blank input fields is much better than to force users to enter nonsense.
- Null values are not the same as the empty string: The empty string is a defined value and poses no problem.

  Allowing the empty string and a null value for the same attribute may make it difficult to distinguish between the two: What does an empty field in a screen mask/form mean?
- Key attributes cannot be optional.

  At least the primary key, which is marked by underlining, must be defined. "Anonymous entities" are not allowed.

## Structured Attributes

- It may be advantageous to treat values for different attributes as a whole (like a record data type).
- A top-down design of attributes could leave the inner structure for later definition (refinement).



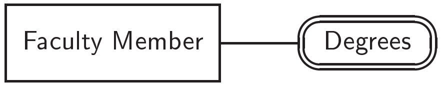

- Non-structured attributes are treated as atomic.
  Their components can only be accessed via programming.
- Structured attributes can be used in keys: Either underline “Name” or the two components “First” and “Last”.





## Multivalued Attributes (1)

- Some attributes may have several values:



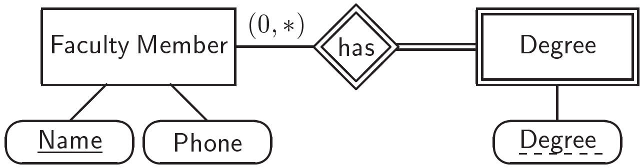

- The attribute is in this case set-valued.
- Multivalued attributes allow the empty set (similar to null).
- This is not the same as a string with all the values concatenated (and maybe separated by some delimiter).
  This would be one possible encoding/translation into the relational model, but not the only one (see below).
- Multivalued attributes cannot be used as keys.





## Multivalued Attributes (2)

- Multivalued attributes can always be transformed into weak entities:




- This is more powerful than the multivalued attribute:
  - Relationships can be defined on the elements of the set.
  - Stricter keys can be defined for the weak entity.
    If an attribute is multivalued and structured, and not all fields are needed for identification, use a weak entity.





## Overview

1. General Remarks on Extended ER-Models
2. Advanced ER-Constructs I: Weak Entities
3. Advanced ER-Constructs II:
   Optional, Multivalued, Structured Attributes
4. Advanced ER-Constructs III: Specialization
5. Miscellaneous Remarks on DB Design

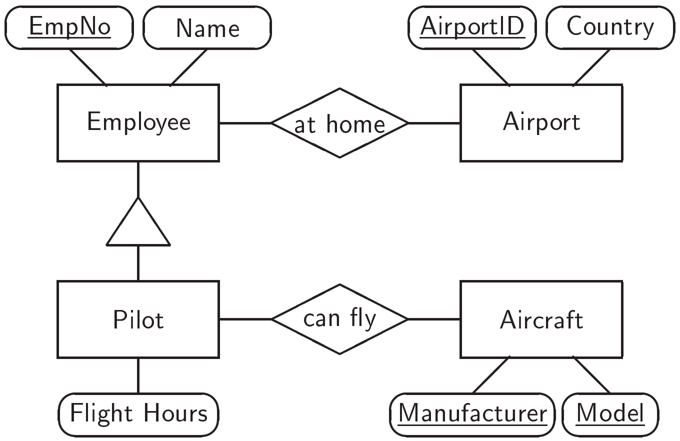
## Specialization (1)

- Two or more entity-types may have attributes or relationships in common.
- Then it might be useful to create a generalized entity type ("superclass"), which contains only the common characteristics, and abstracts from the differences.
- Or, some attributes or relationships may apply to only a subset of the entities. Then creating a specialized entity-type ("subclass") for this set should be considered.
- Inheritance ("is-a" relationships) have proven useful in knowledge representation and object-oriented programming languages.





## Specialization (2)

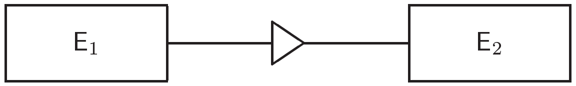







## Specialization (3)

- In this course, the following notation is used for specialization:




- It means that every $E_1$ "is a" $E_2$, i.e. $\mathcal{S}(E_1) \subseteq \mathcal{S}(E_2)$. If the triangle is read as an arrow, it points in the "is a" direction.
- $E_1$ (the subset) automatically inherits all attributes and relationships from $E_2$ (the superset).

  Since the set of pilots is a subset of the set of employees, all attributes defined on employees are defined on pilots in particular. If employees can participate in a relationship, this is especially true for pilots.





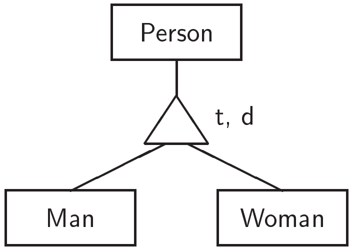
## Specialization (4)

- Since attributes are inherited from the superclass, it is not permissible to define an attribute with the same name in the subclass. A similar rule applies to relationships.
- $E_1$ (the subclass, e.g. pilot) needs no extra key:
  A key of $E_2$ (the superclass) is automatically a key for $E_1$.
- E.g. if employees are uniquely identified by "EmpNo", there especially cannot be two pilots with the same "EmpNo".

  In exceptional cases, the inherited key might be non-minimal. E.g. if employees are identified by job and a number, the job would not be needed for pilots.
- Thus, primary keys (underlined attributes) cannot be defined for subclasses. Alternate keys can be identified (in the text).

# Specialization (5)

- An entity type can have multiple subtypes:




- It should be specified whether the subsets are disjoint (d), i.e. $\mathcal{S}(\text{Man}) \cap \mathcal{S}(\text{Woman}) = \emptyset$, or overlapping (o),
- and whether the specialization is total (t), i.e. $\mathcal{S}(\text{Man}) \cup \mathcal{S}(\text{Woman}) = \mathcal{S}(\text{Person})$, or partial (p).





# Specialization (6)

- Exercise: Suppose there is an entity type for persons at the university, with the subtypes students and employees.
  - Are these subtypes disjoint?
  - What would it mean that the specialization is partial?
- In most object-oriented programming languages, an object needs to have a unique type (i.e. belong to a single most-specific subclass).

  Here, entity-types are not necessarily disjoint. This is sometimes a simplification. In programming languages, for every combination of classes that can have common objects a joint subclass must be introduced (multiple inheritance).





# Specialization (7)

- Subtypes can have their own subtypes.

  Students and employees are both members of a university. Students can in turn be graduate or undergraduate.
- An entity type can be specialized in different ways at the same time (orthogonal issues).

  E.g. employees can be faculty or staff members, and they can participate in a dental insurance plan or not.
- An entity type can be a subtype of multiple supertypes.

  But there must be a common supertype at the end, from which a common key is inherited. Also, each inherited attribute must be declared in a unique superclass.
  Of course, cycles in the subtype relation are excluded.



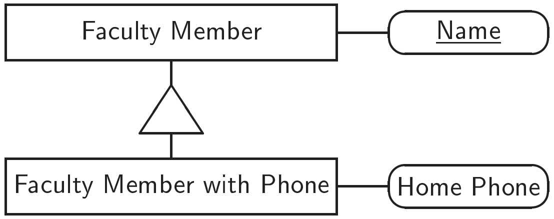


# Specialization and Constraints

- Often, there are constraints related to subtypes.
- E.g., “Employee” might have an attribute “job”, and the subtype “Pilot” might then contain exactly those employees with “job=Pilot” (attribute-defined specialization).
- Such constraints can help for the translation into the relational model.
- A constraint should be specified if the possible values for an inherited attribute are more restricted in the subtype.

## Optional Attributes vs. Specialization

- Optional attributes can be represented with specialization:




- However, if various attributes can be missing, there will be a lot of such subclasses.
- So here a subclass would not be good style.
- However, if there is a group of attributes which must be either all defined or all undefined, a subclass is indicated.





## Generalization / Categorization

- The specialization process starts with the superclass (e.g. Employee), and constructs subclasses (e.g. Pilot).
  Conversely, subclasses could be identified first, e.g. Student and Employee, followed by discovery of common attributes, which results in the superclass "University Member".
- Beginning with the subclasses is called generalization.
  The result is identical to the specialization; it is only a question of the order in which the entity types are introduced.
- Some authors use the term generalization/categorization if the subclasses have keys of their own, and their union should be considered for certain purposes (e.g. a relationship).
  In this course, a key must be defined in the topmost class.





## Formal Graphical Syntax, Ver. 3 (1)

- An ER-Diagram contains boxes, diamonds, ovals (each may have single or double borders), triangles (only with single borders), and interconnecting lines (single and double).
- Interconnecting lines are only allowed between
  - a box and a diamond,
  - a box and an oval,
  - an oval and an oval (the two ovals must be distinct),
  - a diamond and a oval,
  - a triangle and a box.

    A triangle can have connections only on one edge and the opposite corner.





## Formal Graphical Syntax, Ver. 3 (2)

- For diamonds, these constraints must be satisfied:
  - A diamond must have exactly two connecting lines to boxes. There may be any number to ovals.
  - If the diamond has single borders, the connecting lines must be single.
  - If the diamond has double borders, exactly one of the connecting lines must be double, and it must connect to a box with double borders.
- If a box has double borders, it must have at least one double connecting line (which connects to a diamond with double borders).

## Formal Graphical Syntax, Ver. 3 (3)

- A double line may exist only between a box with double borders and a diamond with double borders.
- From each oval, there must be exactly one path (through interconnecting lines and possibly other ovals) to a box or a diamond.

  Of course, only paths count which do not pass through the same node twice.
- For triangles, the following constraints must be satisfied:
  - Every triangle must have at least two connecting lines: One on an edge and one on the opposite corner.





## Formal Graphical Syntax, Ver. 3 (4)

- Constraints on triangles, continued:
  - The lines connecting on the edge must lead to boxes with single borders, the lines on the corner can lead to boxes with single or double borders.
  - The directed graph constructed from the boxes as vertices and the connections via triangles (in the “arrow direction”) as edges is acyclic.
  - If in this graph, from a box $A$ two boxes $B$ and $C$ are reachable, then there is a box $D$ such that $D$ is reachable from (or equal to) $B$ and $C$.





## Formal Graphical Syntax, Ver. 3 (5)

- Boxes, diamonds, and ovals are each labelled by a string.

  The string is written into the construct.
- The names of boxes must be unique in the entire diagram.

  I.e. you cannot have two boxes with the same label.
- The names of ovals must only be unique for a single box or diamond.

  I.e. two ovals with the same name which are linked to the same box (or the same diamond) (maybe indirectly via other ovals) are forbidden. It is also forbidden that ovals with the same name are linked to boxes $A$ and $B$, and there is a path (via triangles in arrow direction and possibly other boxes) from $A$ to $B$.





## Formal Graphical Syntax, Ver. 3 (6)

- Diamonds must be uniquely identified by name and connection.

  I.e. two diamonds which are linked to the same two boxes and have the same name are excluded. In addition, it is also forbidden that there is one diamond between $A$ and $C$ and one diamond between $B$ and $D$, both labelled with the same name, and there are connections via traingles between $A$ and $B$ and between $C$ and $D$.

## Formal Graphical Syntax, Ver. 3 (7)

- The string in ovals may be underlined if each of the following is true:
  - The oval is directly connected to a box with single borders.
  - The oval has single borders and is not connected (maybe indirectly via other ovals) to an oval with double borders.
  - The box is not connected to an edge of a triangle (it may be connected to a corner).
- The string in ovals connected to boxes with double lines may be dotted underlined if the oval has single borders and is not connected (maybe indirectly via other ovals) to an oval with double borders.





## Formal Graphical Syntax, Ver. 3 (8)

- Every box with single borders that is not connected to an edge of a triangle must be connected to at least one oval with an underlined string.
- The lines which connect a diamond and a box can be labelled with a string of the form $(n, m)$ satisfying the following conditions:
  - $n$ is a non-negative number (i.e. $n \geq 0$).
  - $m$ is a number or the symbol "$*$".
  - If $m$ is a number, then $n \leq m$ holds.
- Near to a triangle, at most one of the following strings may be written: "(t,d)", "(t,o)", "(p,d)", "(p,o)".





## Overview







## Sources for Information

- Interviews
- Analysis of existing forms.

  Completed forms are more useful than blanks.
- Analysis of reports which should be generated from the DB.
- Analysis of existing files.
- Analysis of existing manuals, descriptions, documentations (e.g. information for new employees, business reports).
- Partial reuse of published solutions ("reference models").
- Experiences of other companies (consultants).
- Software like SAP/R3 comes with a database schema.

## Validation

- Use example scenarios. Review them with future users.
- If names of entities and relationships are chosen rigorously, explanatory natural language sentences may be produced automatically.
  - E.g. “Each and every MUSIC PIECE may be COMPOSED BY one and only one COMPOSER”.
  - Future users may check such sentences.
- Cross-check database design with program specification.
  - Does the database provide all necessary information?
  - Is every entity type created/retrieved/updated/deleted by some program (CRUD cycle)?





## View Integration

- Often, it is impossible to create the complete ER-Schema in one step, because this would be too large.
- First create small ER-schemas for each application or user. I.e. start not with the conceptual schema level, but with external views.
- After that, integrate these schemas to get the complete enterprise data model.

  This is not simple: There may be naming conflicts (synonyms, homonyms), type conflicts (the same thing as an entity and as an attribute), domain conflicts (different data types or units of measure), and conflicts among constraints.





## Information Needed for Physical Design

Estimation of the size:

- How many entities are there of each type?
- What is the expected growth rate?
- After what time can entities be deleted from the database?
- How long are the string valued attributes typically?
- How large is the percentage of null values?

Estimation of needed performance:

- How often each application will be executed?
- Which applications require quick response?

  Which can be executed overnight?





## Features of Design Tools

- Repository: Database with all design information.
- Graphical Editor for ER-diagrams.
- Version control, views, structuring of large schemas.
- Style and consistency checkers.
- Automatic/Semiautomatic generation of relational tables.
- Automatic/Semiautomatic generation of form-based application programs for entering/viewing data.
- Automatic generation of triggers for integrity constraints.
- Reverse engineering tools, data dictionary browsers.
- Integration with business process models etc.